B.M.S. COLLEGE OF ENGINEERING BENGALURU
Autonomous Institute, Affiliated to VTU

OOMD Mini Project Report

Blog Application Featuring Clean Architecture

Submitted in partial fulfillment for the award of degree of

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by:

Bramha Anilkumar Bajannavar 1BM23CS071

Daivya Priyankkumar Shah 1BM23CS084

Hitesh Sharma 1BM23CS114

Department of Computer Science and Engineering
B.M.S. College of Engineering
Bull Temple Road, Basavanagudi, Bangalore 560 019
2025-26

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, Bramha Anilkumar Bajannavar(1BM23CS071), Daivya Priyankkumar Shah(1BM23CS084) and Hitesh Sharma(1BM23CS114) students of 5th Semester, B.E, Department of Computer Science and Engineering, BMS College of Engineering, Bangalore, hereby declare that, this OOMD Mini Project entitled "Blog Application Featuring Clean Architecture" has been carried out in Department of CSE, B.M.S. College of Engineering, Bangalore during the academic semester August 2025- December 2025. I also declare that to the best of our knowledge and belief, the OOMD mini Project report is not from part of any other report by any other students.

Signature of the Candidate

Bramha Anilkumar Bajannavar(1BM23CS071)

Daivya Priyankkumar Shah(1BM23CS084)

Hitesh Sharma(1BM23CS114)

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the OOMD Mini Project titled "**Blog Application Featuring Clean Architecture**" has been carried out by Bramha Anilkumar Bajannavar(1BM23CS071), Daivya Priyankkumar Shah(1BM23CS084) and Hitesh Sharma(1BM23CS114) during the academic year 2025-2026.

Signature of the Faculty in Charge (Sonika Sharma D)

Table of Contents

Chapter 1: Problem Statement

With the rapid growth of digital content creation, blogging has become one of the most widely used methods for sharing information, opinions, and personal experiences. However, many existing blog applications face significant limitations such as poor architectural structure, difficulty in scaling, inconsistent data handling, and lack of offline functionality. These issues lead to tightly coupled components, increased maintenance overhead, and challenges in integrating new features. Users also expect smooth interaction, fast responses, and reliable data synchronization, which many traditional blog systems fail to deliver.

The primary problem identified is the absence of a well-structured, maintainable, and scalable blog application that separates concerns clearly and ensures robust data flow. Without a clean architecture, the application becomes harder to test, debug, and extend. Developers need a system where presentation logic is independent of business logic, and business logic is independent of data sources, enabling flexibility in choosing or switching technologies without affecting other layers.

To address this, the project aims to build a blog application that supports core functionalities such as creating posts, editing posts, viewing posts, uploading images, and authenticating users. The system must also ensure smooth data handling across both online and offline conditions, preventing data loss when network connectivity is unstable.

The blog application must allow users to register, log in, and manage their accounts, as well as create, edit, delete, and view blog posts with optional image uploads and remote data storage while also supporting offline drafts through local caching. The system should retrieve posts reliably from the backend, maintain user authentication securely, and ensure smooth interaction through proper state management. In addition to these functional requirements, the application must also satisfy key non-functional requirements such as maintaining a modular and scalable architecture, providing a responsive and user-friendly interface, ensuring consistent performance even under unstable network conditions, and supporting easy testing, debugging, and future feature expansion.

To meet these needs, the application adopts a Clean Architecture approach, which organizes the project into Presentation, Domain, and Data layers. This separation improves clarity, reduces dependency issues, and enhances testability.

Overall, this project solves the need for a well-architected, robust, and scalable blog application by combining clean design principles with modern development practices, ensuring both maintainability for developers and reliability for end users.

Chapter 2: Software Requirement Specification

1. Introduction

1.1 Purpose

The purpose of this document is to define and describe the requirements of the Blog Application developed using the Clean Architecture pattern. It establishes a clear understanding of what the system aims to achieve and outlines the functional and non-functional expectations. This ensures that developers, testers, evaluators, and stakeholders have a common reference point throughout the development lifecycle.

1.2 Intended Audience

This SRS is intended for developers working on the system, testers validating the application, project evaluators reviewing the system's design and functionality, and any stakeholder involved in understanding the technical and behavioral aspects of the application. It provides clarity for both technical and non-technical readers regarding how the system is structured and what it is expected to accomplish.

1.3 Scope

The scope of this system includes providing users with the ability to create, edit, delete, and view blog posts with image support, user authentication, and offline drafting. The application is designed for mobile platforms and aims to offer a smooth user experience with robust data handling. The scope also includes ensuring a scalable and maintainable architecture, enabling future enhancements without affecting core functionality.

1.4 Definitions and Acronyms

- Post: A blog entry created by a user.
- Draft: A locally saved post when offline.
- Repository: Module handling data retrieval/storage.
- BLoC: Business Logic Component for state management.

2. Overall Description

2.1 Product Perspective

The Blog Application is an independent system built using Flutter and structured using Clean Architecture. It interacts with external services such as Supabase for authentication and remote storage, while also utilizing local storage mechanisms for offline drafts. The system follows a layered architecture to ensure separation of concerns.

2.2 Product Features

The application supports user registration, login, blog creation, editing, deletion, image uploads, and offline drafting. Posts are displayed in a feed format, and user sessions are maintained securely.

2.3 User Characteristics

The primary users are content creators who want a simple, intuitive interface for blogging. No advanced technical knowledge is required, only basic familiarity with mobile applications.

2.4 Constraints

The system depends on network availability for uploading posts and authentication. Device storage limitations may affect caching. Clean Architecture principles must be followed throughout development.

3. Specific Requirements

3.1 Functional Requirements

1. The system must allow users to register and log in securely.
2. The system must allow users to create, edit, delete, and view blog posts.
3. Users must be able to upload images along with their posts.
4. The system must store posts on a remote backend and retrieve them when requested.
5. The system must support offline drafting by saving posts locally when internet is unavailable.
6. The system must display all posts in a structured feed format.
7. The system must maintain user authentication status across sessions.
8. The system must validate input fields before submitting any data.

3.2 Non-Functional Requirements

1. The system must follow a modular Clean Architecture structure for maintainability.
2. The UI must be responsive, user-friendly, and consistent across devices.
3. The system must handle network failures gracefully without data loss.
4. State management must ensure predictable and stable behavior.
5. The system should support future feature expansion with minimal changes to existing code.
6. Performance must remain smooth with minimal loading times.

7. The system must ensure secure handling of authentication and user data.

4. Appendices

Appendix A – Technologies Used

- Flutter
- Clean Architecture
- BLoC State Management
- Supabase Backend
- Local Cache Storage

Appendix B – Definitions

- Post
- Draft
- Repository
- State Management

Appendix C – Assumptions

- Users have internet access for posting and authentication.
- Mobile device storage is available for caching.
- Users understand basic navigation interactions.

Chapter 3: Class Modeling

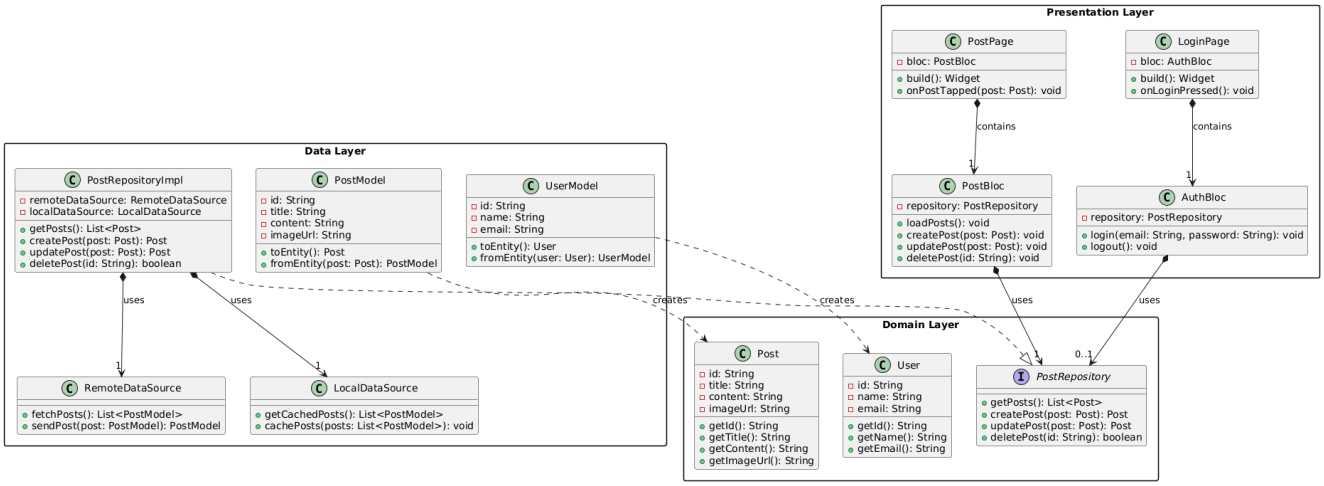

Figure 3.1: Advanced Class Diagram

Class Descriptions:

1. Domain Layer
    - Post: Represents a blog post containing id, title, content, and optional image information.
    - PostRepository: Defines the abstract operations for creating, retrieving, updating, and deleting posts.
    - User: Represents a system user with basic identity details like id, name, and email.

2. Data Layer
    - PostModel: Model version of a Post used for converting between domain objects and data sources.
    - UserModel: Data model for User used in storage and network communication.
    - PostRepositoryImpl: Implements PostRepository by coordinating both remote and local data sources.
    - RemoteDataSource: Handles fetching and sending post data to the remote backend server.
    - LocalDataSource: Manages local caching of posts for offline access and faster retrieval.

3. Presentation Layer
    - PostBloc: Controls and manages all post-related UI logic and state updates.
    - AuthBloc: Manages user login and logout operations and maintains authentication state.
    - PostPage: UI screen that displays posts and interacts with the PostBloc.
    - LoginPage: UI screen for user authentication that communicates with the AuthBloc.

Chapter 4: State Modeling

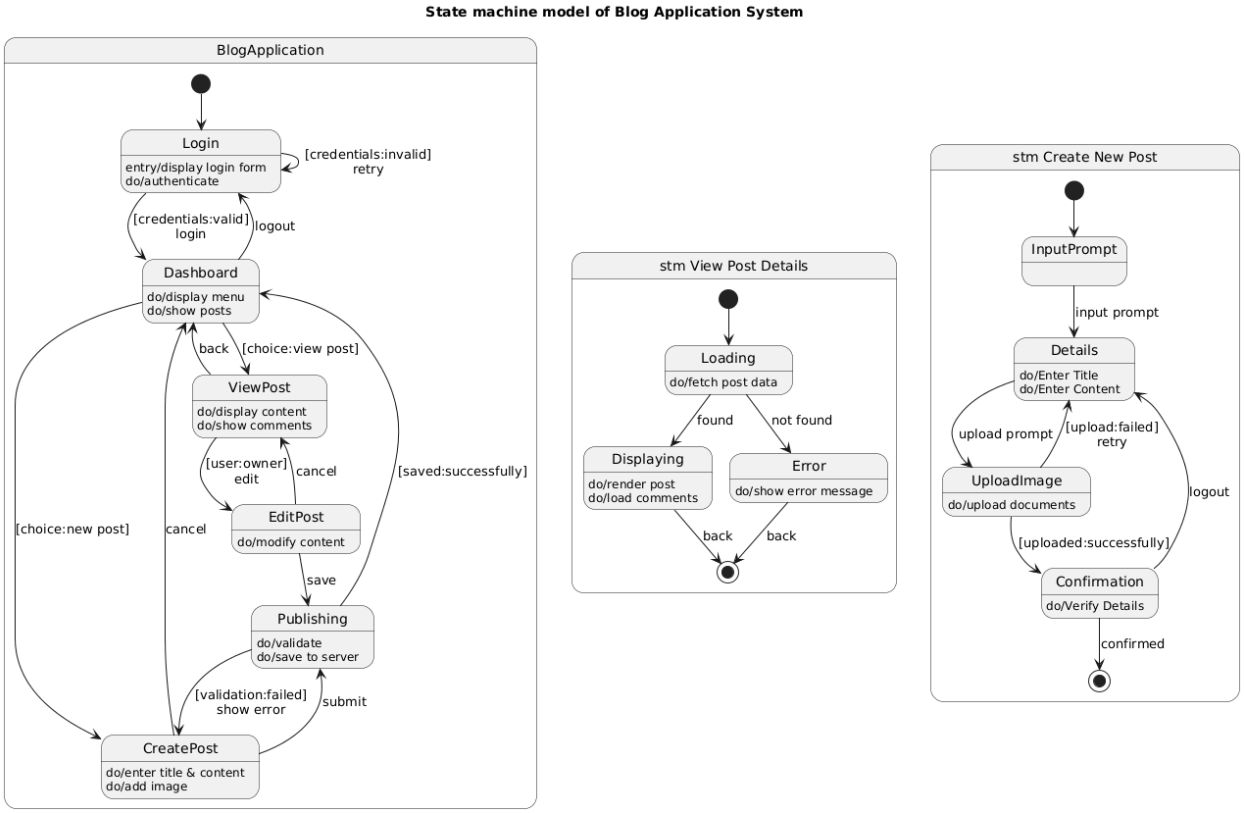

Figure 4.1: Advanced State Diagram

State Description:

1. Blog Application
    - Login: Handles user authentication by displaying the login form and validating credentials.
    - Dashboard: Main screen that shows available actions like viewing posts or creating a new one.
    - CreatePost: Allows the user to enter post content and optionally upload an image.
    - ViewPost: Displays the selected blog post along with its comments.
    - EditPost: Lets the post owner modify the existing post content.
    - Publishing: Validates and saves the post to the server before completing the action.
    - BlogApplication (Composite State): Represents the entire app workflow, containing login, dashboard, and post-related states.

2. View Post Details

- Loading: Fetches the selected post data from the server or local cache.
- Displaying: Renders the post and loads associated comments for reading.
- Error: Shows an error message if the post cannot be loaded.
- ViewPostDetail (Composite State): Represents the complete flow of opening and viewing a single post.

3. Create New Post
    - InputPrompt: Initial state prompting the user to begin entering post details.
    - Details: State where the user inputs the post title and content.
    - UploadImage: Handles uploading of an optional image for the post.
    - Confirmation: Displays a summary of the entered post details for final user confirmation.
    - CreateNewPost (Composite State): Covers the entire workflow of drafting, uploading, and confirming a new post.

Event Description:

- [credentials:valid] login: Allows the system to transition from the Login state to the Dashboard once authentication succeeds.
- [credentials:invalid] retry: Keeps the user in the Login state when incorrect credentials are entered.
- [choice:new post]: Moves the user from the Dashboard to the CreatePost state to begin writing a new blog.
- [choice:view post]: Opens the selected post from the Dashboard and transitions into the ViewPost state.
- Submit: Triggers validation and publishing when the user finishes creating a post.
- Save: Saves changes made in EditPost and transitions the system to the Publishing state.
- Logout: Returns the user to the Login state from any authenticated state.
- Found: Indicates that a post has been successfully retrieved in the ViewPostDetail machine.
- not found: Signals an error when the requested post cannot be fetched.
- [uploaded:successfully]: Moves the CreateNewPost workflow into the Confirmation state after successful image upload.

Chapter 5: Interaction Modeling

Use Case Diagram:

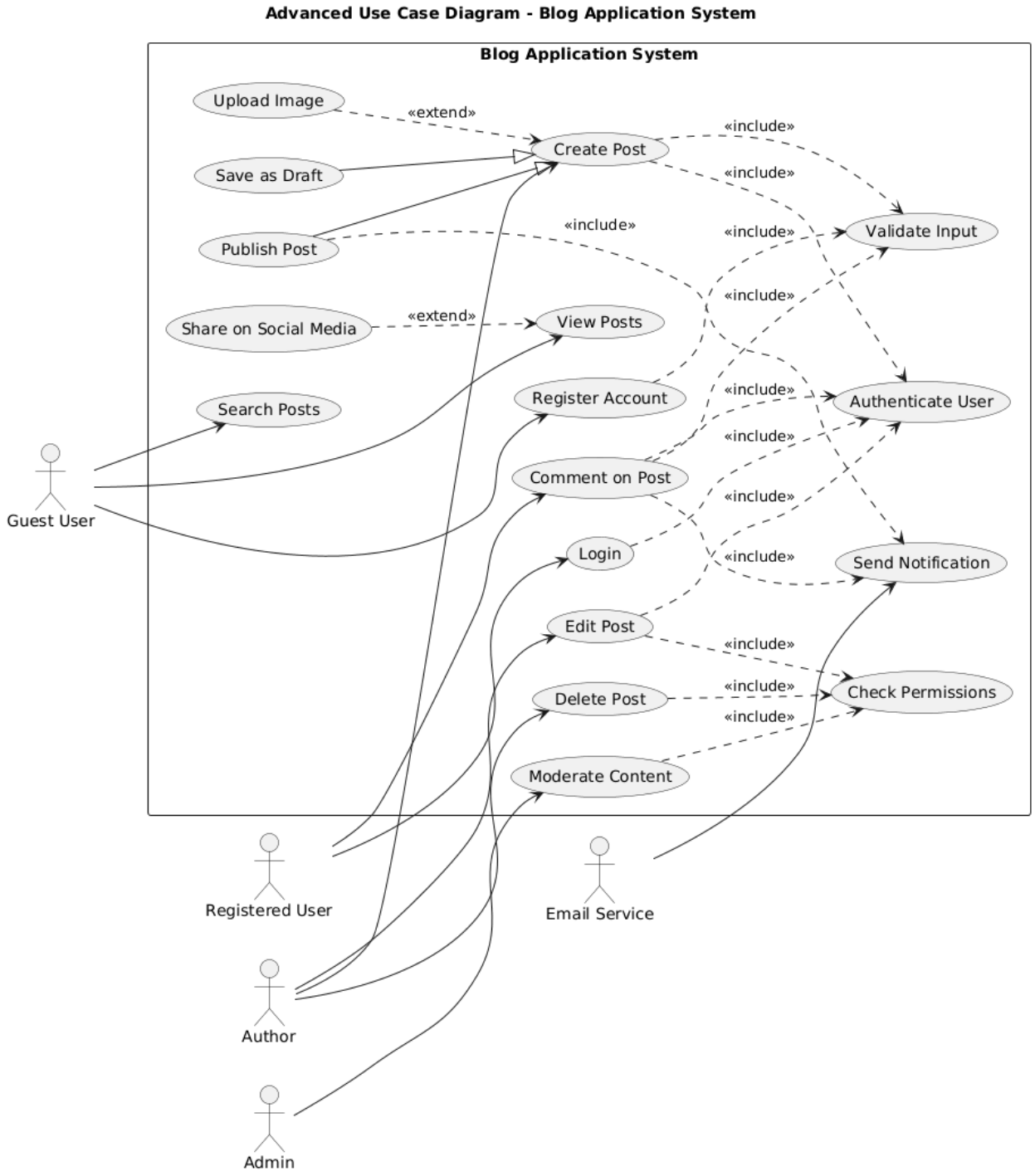

Figure 5.1: Use Case Diagram

Relevance of Actors:

- Guest User: A user who has not logged in; can browse posts, search content, and register for an account.
- Registered User: A logged-in user who can comment on posts and access personalized features.
- Author: A registered user with permissions to create, edit, and delete their own blog posts.
- Admin: Manages the platform by moderating inappropriate content and ensuring community guidelines are maintained.
- Email Service: An external system responsible for sending notifications like registration confirmation or post alerts.

Relevance of Use Cases:

- View Posts: Allows any user to browse the available blog posts.
- Search Posts: Lets users quickly find posts based on keywords.
- Register Account: Enables new users to create an account for personalized access.
- Login: Authenticates the user before accessing restricted features.
- Comment on Post: Allows registered users to comment and engage with posts.
- Create Post: Permits authors to write and submit new blog content.
- Edit Post: Allows authors to update or revise their existing posts.
- Delete Post: Enables authors to remove a post they own from the system.
- Moderate Content: Allows admins to review and take action on inappropriate content.
- Upload Image: Lets authors attach images to their posts during creation.
- Publish Post: Makes the post publicly visible after creation or editing.
- Save as Draft: Stores incomplete posts so the author can continue later.

Sequence Diagram:

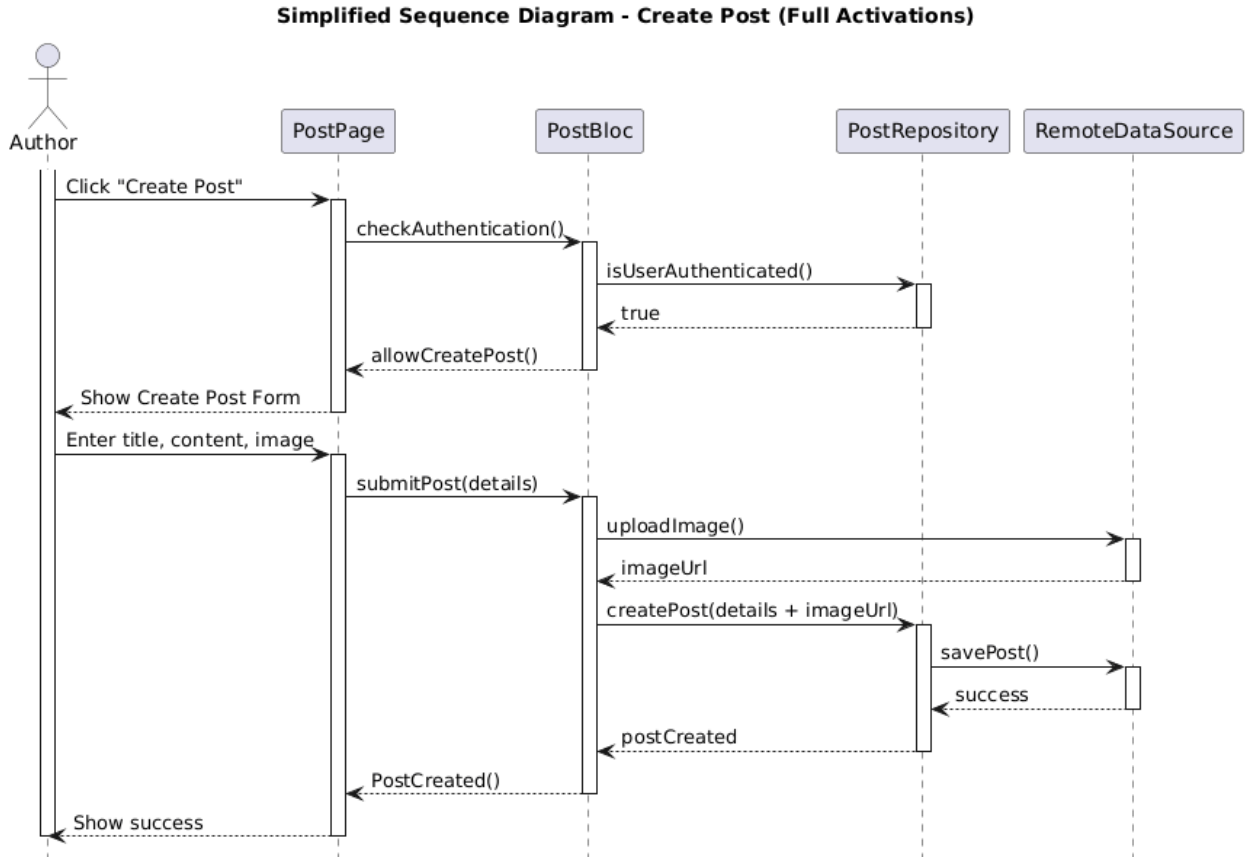

Figure 5.2: Sequence Diagram

Brief Description:

The Create Post sequence diagram illustrates the interaction flow between the user and the system components during the creation of a new blog post. The process begins when the Author initiates the action by selecting the “Create Post” option. The UI first communicates with the PostBloc to verify whether the user is authenticated. The Bloc then checks this through the Repository, which confirms authentication and allows the user to proceed with the post creation form.

Once the Author enters the post details, the UI sends the submitted information to the Bloc. The Bloc first uploads the selected image to the RemoteDataSource, which returns the generated image URL. Using this complete post data, the Bloc requests the Repository to create the post. The Repository communicates with the RemoteDataSource to store the post on the server, and once successful, the confirmation is passed back through the Bloc to the UI. Finally, the UI notifies the Author with a success message, completing the post creation workflow.

Activity Diagram:

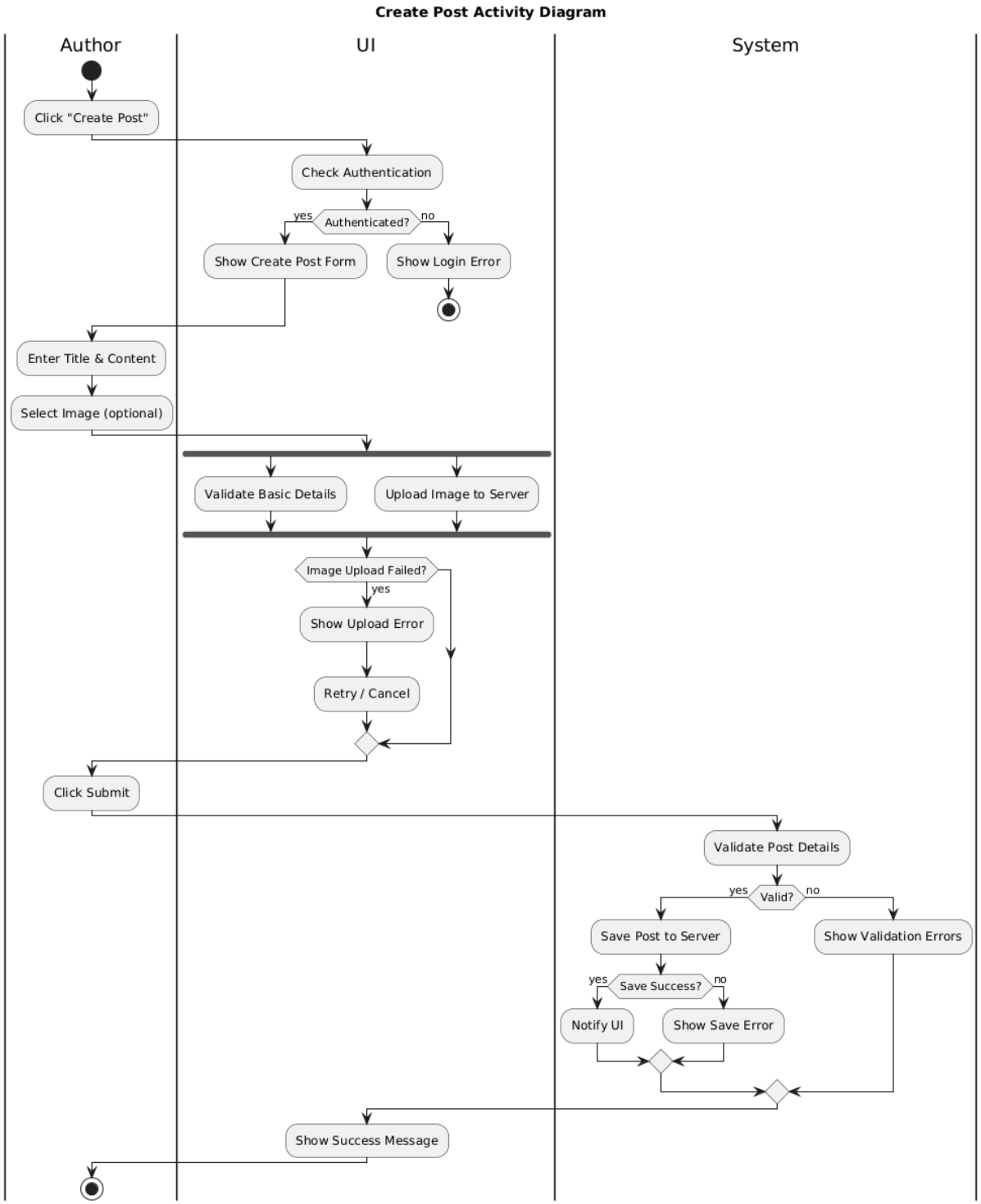

Figure 5.3: Activity Diagram

Brief Description:

The activity diagram models the complete “Create Post” workflow by separating actions into swimlanes for the Author, UI, and System. The process begins when the Author initiates post creation, and the UI first verifies authentication before showing the post creation form. Once authenticated, the Author enters the post title, content, and optionally selects an image. At this point, the workflow splits into two parallel activities: the UI validates the basic details while the System (via the UI) handles the image upload to the server. This concurrent processing improves efficiency by allowing the form validation and image upload to happen simultaneously.

After both parallel tasks merge, the Author submits the post. The System then performs a complete validation check and attempts to save the post to the server. If the operation succeeds, the UI returns a success message to the Author; otherwise, appropriate error messages are shown for validation, upload, or saving failures. The swimlane structure clearly illustrates how responsibilities are distributed across different components, while the fork–join region shows that certain tasks can proceed concurrently to optimize the user experience.

Chapter 6: UI Design with Screenshots

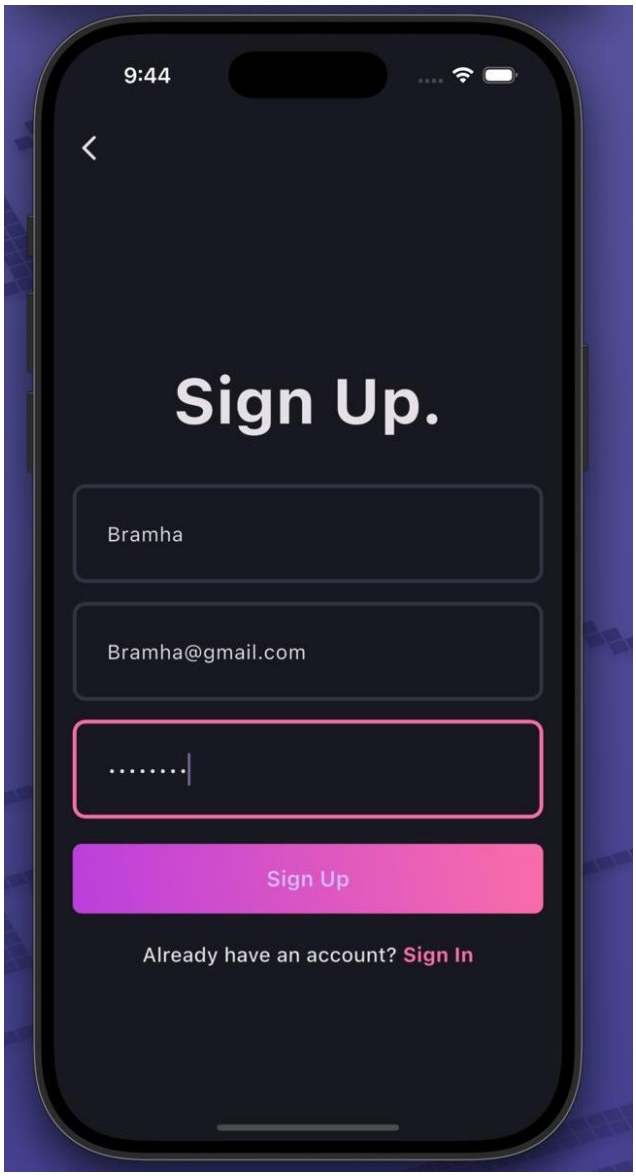

Figure 6.1: Sign Up Page

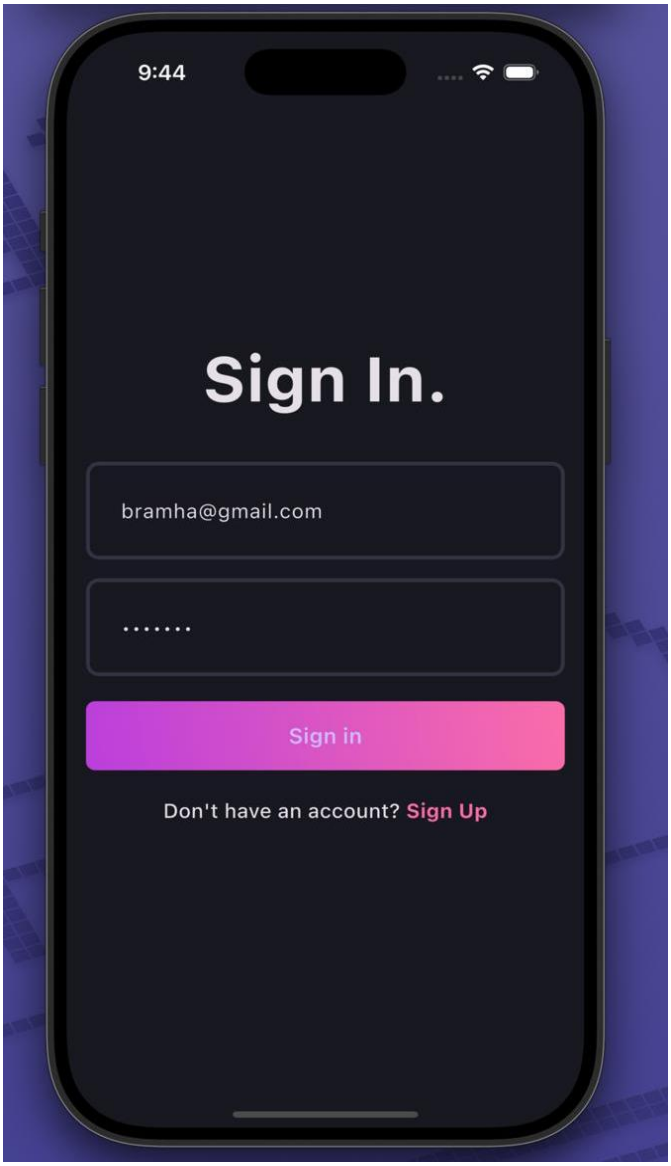

Figure 6.2: Sign In Page

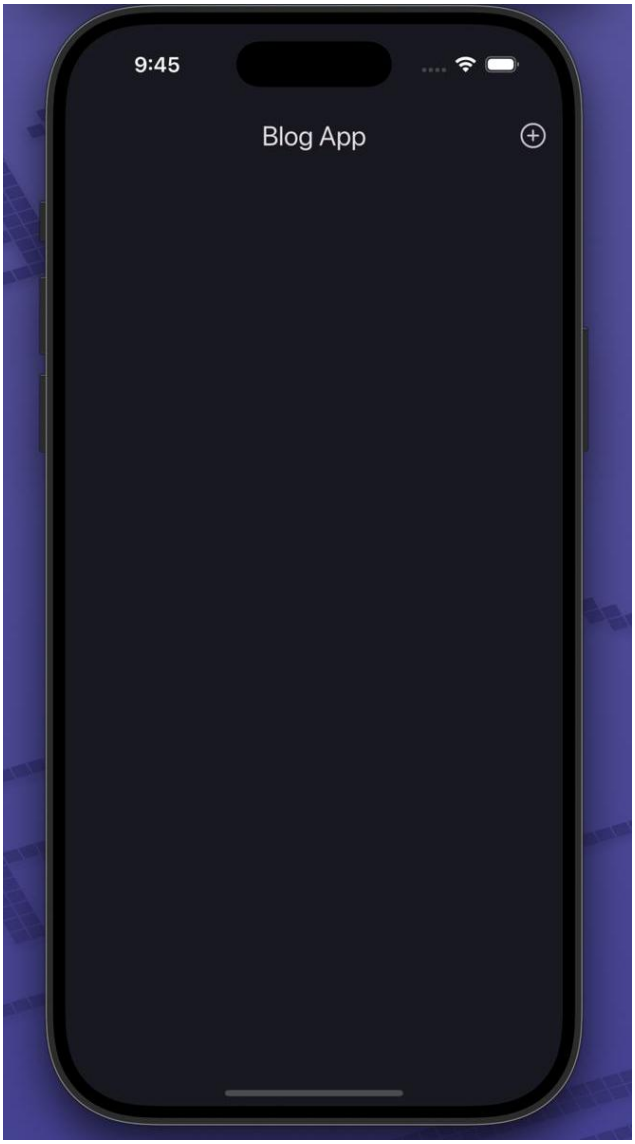

Figure 6.3: Home Page

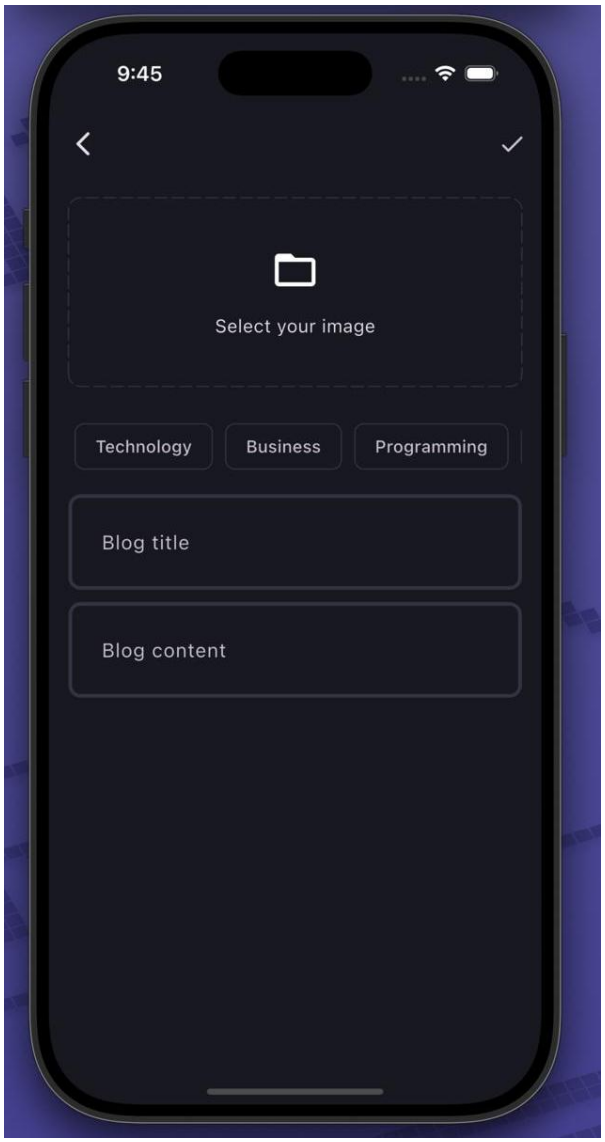

Figure 6.4: Create Blog Page

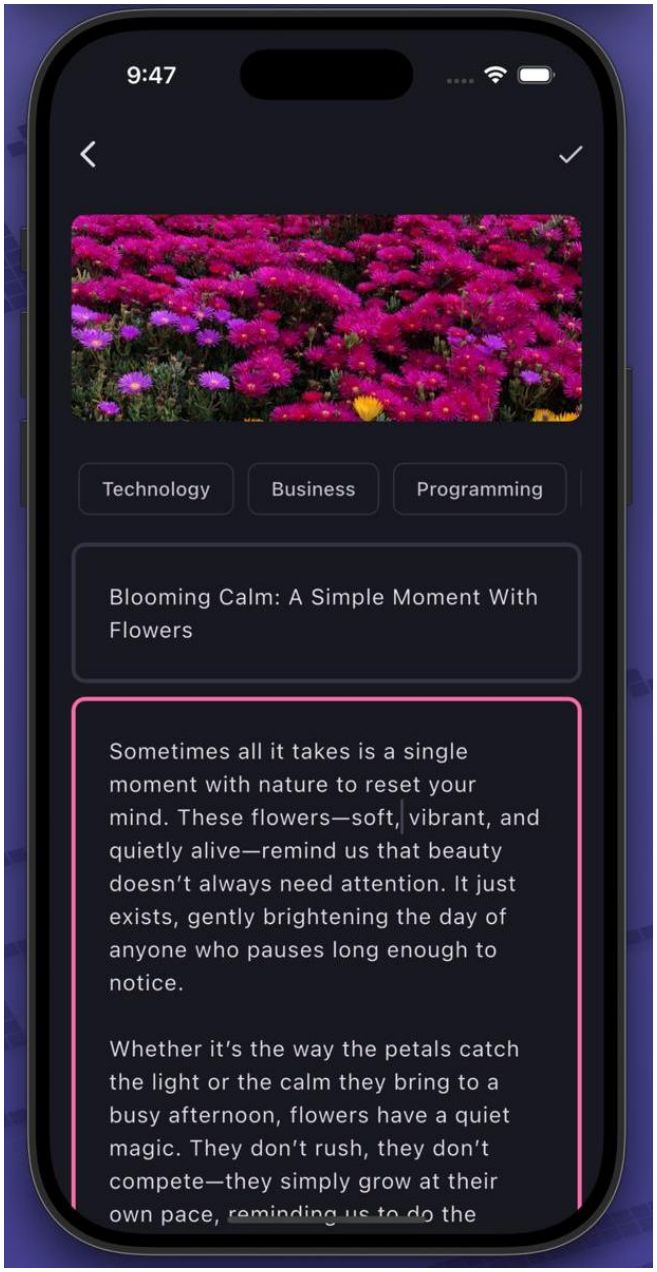

Figure 6.5: Blog with added details

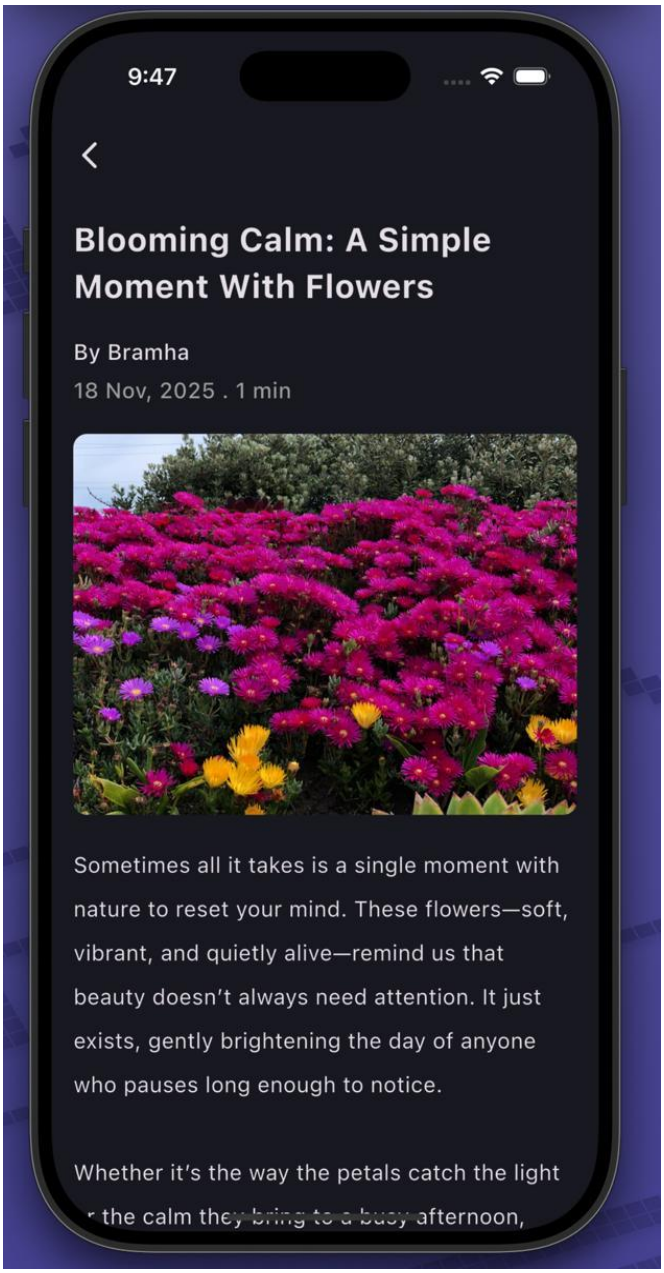

Figure 6.6: View Blog